**Exception Handling in Java**

The **exception handling** is one of the powerful mechanisms to handle the runtime errors so that the normal flow of the application can be maintained.

In java, exception occurs during the execution of the program, and it disrupts the normal flow. Exception can occur because of various reasons, such as invalid input, file not found.

When an exception occurs, it is typically represented by an object of a subclass of the java.lang.Exception class. And throws an exceptions ClassNotFoundException, IOException, SQLException, RemoteException, etc.

**Why to handle exceptions?**

To maintain the normal flow of the application, hence we need to handle the exception.

**Example: Let’s say we are executing a test for 10 data set and at second test data set we got exception it will throw the exception and stop the execution flow, hence to continue the flow exception handling is important.**

**Types of Java Exceptions**

In Java, exceptions are categorized into two main types: checked exceptions and unchecked exceptions.

Checked Exception 2. Unchecked Exception

**1. Checked Exceptions (Compile Time)**

Checked exceptions are the exceptions that are checked at compile-time. This means that the compiler verifies that the code handles these exceptions either by catching them or declaring them in the method signature using the throws keyword. Examples of checked exceptions include:

* **IOException:** An exception is thrown when an input/output operation fails, such as when reading from or writing to a file.
* **SQLException:** It is thrown when an error occurs while accessing a database.
* **ParseException:** Indicates a problem while parsing a string into another data type, such as parsing a date.
* **ClassNotFoundException:** It is thrown when an application tries to load a class through its string name using methods like Class.forName(), but the class with the specified name cannot be found in the classpath.

**2. Unchecked Exceptions (Runtime Exceptions)**

Unchecked exceptions, also known as **runtime exceptions**, are not checked at compile-time. These exceptions usually occur due to programming errors, such as logic errors or incorrect assumptions in the code. They do not need to be declared in the method signature using the throws keyword, making it optional to handle them. Examples of unchecked exceptions include:

* **NullPointerException:** It is thrown when trying to access or call a method on an object reference that is null.
* **ArrayIndexOutOfBoundsException:** It occurs when we try to access an array element with an invalid index.
* **ArithmeticException:** It is thrown when an arithmetic operation fails, such as division by zero.
* **IllegalArgumentException:** It indicates that a method has been passed an illegal or inappropriate argument.

**Java Exception Keywords**

Java provides five keywords that are used to handle the exception. The following table describes each.

|  |  |
| --- | --- |
| **Keyword** | **Description** |
| **Try** | **The "try" keyword is used to specify a block where we should place an exception code(code logic that may throws an exception). It means we can't use try block alone. The try block must be followed by either catch or finally.** |
| **Catch** | **The "catch" block is used to handle the exception. It must be preceded by try block. It can be followed by finally block later.** |
| **finally** | **The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not.** |
| **throw** | **The "throw" keyword is used to throw an exception.** |
| **throws** | **The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature.** |
| **Keyword** | **Description** |
| Try | The "try" keyword is used to specify a block where we should place an exception code. It means we can't use try block alone. The try block must be followed by either catch or finally. |
| Catch | The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later. |
| finally | The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not. |
| throw | The "throw" keyword is used to throw an exception. |
| throws | The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature. |

1. //Java Program to understand the use of exception handling in Java
2. **public** **class** Main{
3. **public** **static** **void** main(String args[]){
4. **try**{
5. **int** data=100/0;
6. }**catch**(ArithmeticException e){System.out.println(e);}
7. *//rest code of the program*
8. System.out.println("rest of the code...");
9. }
10. }

The try-catch Block

One of the primary mechanisms for handling exceptions in Java is the try-catch block. The try block contains the code that may throw an exception, and the catch block is used to handle the exception if it occurs. Here's a basic example:

1. **try** {
2. *// Code that may throw an exception*
3. } **catch** (ExceptionType e) {
4. *// Exception handling code*
5. }

The finally Block

In addition to try and catch, Java also provides a finally block, which allows you to execute cleanup code, such as closing resources, regardless of whether an exception occurs or not. The finally block is typically used to release resources that were acquired in the try block. Here's an example:

1. **try** {
2. *// Code that may throw an exception*
3. } **catch** (Exception e) {
4. *// Exception handling code*
5. } **finally** {
6. *// Cleanup code*
7. }